Express Mail Label No. EL818623630US

HARDWARE EFFICIENT HANDLING OF INSTRUCTION EXCEPTIONS TO LIMIT ADVERSE IMPACT ON PERFORMANCE

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] Not applicable.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] Not applicable.

BACKGROUND OF THE INVENTION

Field of the Invention

[0003] The present invention generally relates to the interpretation and execution of software instructions by a processor in a computer system. More particularly, the present invention relates to the handling of program instructions for which an exception has occurred. Still more particularly, the present invention relates to an exception handler for a processor that operates more efficiently by dividing excepted instructions into different categories.

Background of the Invention

[0004] A computer system includes a number of components with specialized functions that cooperatively interact to produce the many effects available in modern computer systems. The ability of these various components to exchange data and other signals is vital to the successful operation of a computer system. Typically, components interact by reading or writing data or instructions to other components in the system.

[0005] Computer systems typically include a processor (or CPU), random access memory (RAM), and certain peripheral devices such as a floppy drive, a keyboard and a display. These components typically couple together using a network of address, data and control lines, commonly referred to as a "bus." As computer technology evolved, it became common to connect additional peripheral devices to the computer through ports (such as a parallel port or a serial port), or by including the peripheral device on the main system circuit board (or "motherboard") and connecting it to the system bus.

[0006] The computer operates by having data flow through the system, with modification of the data occurring frequently. Traditionally, the CPU controls most activities in the computer system. The CPU supervises data flow and is responsible for most of the high-level data modification in the computer. In addition, the CPU receives signals from the peripheral devices, reads and writes data to memory, processes data, and generates signals controlling the peripheral devices. The CPU is often referred to as the "brain" of the computer system.

[0007] The CPU is a device that operates according to instructions programmed by a designer or programmer. Thus, every operation that the CPU performs is based on one or more programmed instruction. In normal operation, the CPU performs many operations on an instruction, three of which are: (1) it fetches the instruction; (2) it decodes the instruction; and (3) it executes the instruction. In a single "pipeline" CPU architecture, each of these different operations is performed sequentially. To improve CPU performance, modern processor architectures often include a plurality of instruction pipelines to enable the CPU to perform operations on multiple instructions in parallel. To further improve CPU performance, pipelined processors often attempt to predict which instructions should be fetched, decoded and perhaps even executed, even before it is certain that the instruction forms part of the program flow. These predictions typically are performed pursuant to parameters programmed in a prediction algorithm that attempt to determine which branch the program will follow. Thus, in modern CPU designs, the CPU may operate on multiple instructions in different pipelines, and may begin operating on instructions that ultimately are not part of the program.

[0008] During normal CPU operation, any error in fetching, decoding, executing or otherwise processing an instruction is referred to as an "exception." Put simply, an excepted instruction is an instruction that cannot be properly executed by the processor. Thus, for example, if an instruction cannot be accurately decoded, or if an instruction code is not recognized, that is an exception event. Another type of exception occurs when the CPU performs an erroneous prediction of the instruction flow. Thus, if the program branches differently than that predicted by the prediction algorithm, then instructions may be fetched and decoded which are unnecessary to proper CPU operation. The initial divergence from the program flow is typically referred to as an exception.

[0009] As processors become faster, they are designed to execute more instructions in parallel. Every instruction has the possibility to except, and thus, as the CPU operates on more instructions, the greater the number of exceptions that will arise. In addition, exceptions can be detected at various stages of the pipeline. As the pipeline becomes longer and wider, more stages or ports are created from which exceptions may be issued. As a further consequence of handling a multiplicity of instructions in parallel, the processor must look further ahead the program flow to fetch and decode instructions that may be needed. The further the CPU must predict ahead, the greater the likelihood that a misprediction will occur, and the more instructions that must be flushed when a misprediction is discovered. It is generally desirable for a CPU to recognize an exception as early as possible, and where necessary, to take corrective steps to handle the exception.

[0010] Thus, modern CPUs must handle an increasingly large number of exceptions. In conventional CPU designs, all exceptions are handled by a single exception handler, regardless of the type of exception. Thus, all exceptions are routed to the exception handler, which typically is implemented on-board the processor chip. The exception handler operates by sorting through the exceptions and selecting an exception to resolve. If the exception was the result of a misprediction, the exception handler would recognize the exception as a misprediction, and would direct the instruction pointer to branch to the appropriate instruction to track the actual program flow. Typically, the exception handler is designed to select the oldest exception to analyze and correct. The oldest exception is the earliest instruction in the program flow. Thus, assume a program is executing with instruction A, B, C, D E and F in sequence. Because most conventional processors fetch instructions in-order, the CPU may fetch instruction B at time *t*, and may fetch instruction F at time *t + 4*. Because of the out-of-order nature of certain processors, the CPU may actually execute instructions B and F at the same time, although they were brought into the CPU at the different times. If instructions B and F both were found to have an exception, the CPU would treat the exception created by instruction B as older than the exception created by instruction F. Because instruction B happened earlier in the program flow, the exception handler would first analyze instruction B for problems. By handling the oldest exception first, the exception handler hopefully will focus its attention on exceptions that are in the correct path of the program, and will not spend an excessive amount of time handling exceptions that are in a mispredicted path. Because instruction B was a misprediction, instruction F would be flushed and would not require resolution.

[0011] As noted above, as modern processor designs operate on a greater number of instructions, more exceptions arise. To handle this contingency, exception handlers in modern processor designs have become increasingly large to enable the exceptions to be queued while they await resolution. Thus, the amount of time to resolve a given exception has increased because of the size of the exception queue, and the many types of exceptions that must be addressed. Thus, while CPU designs have become increasingly fast, exception handlers have become bigger and slower.

[0012] It would be advantageous if a new architecture and method was created to handle exceptions more quickly. In particular, it would be advantageous if certain critical exceptions could be resolved in more expeditious fashion to minimize the latency of the processor, and to achieve greater processor efficiency. Despite these apparent advantages, to date no one has developed an exception handler for a processor that resolves these issues.

BRIEF SUMMARY OF THE INVENTION

[0013] The present invention solves the deficiencies of the prior art by classifying excepted instructions into different categories, to thereby permit critical exceptions to be handled more expeditiously. In particular, the present invention includes two or more exception handlers in the processor circuitry. One exception handler preferably receives critical exceptions, while the other exception handler receives non-critical exceptions.

[0014] According to a preferred embodiment, an exception handler comprises two sections with streamlined operation. The implementation of two exception handling sections permits exceptions to be split into two classes. Exceptions that are performance critical are routed to one section of the exception handler, and exceptions that are not performance critical are routed to the other section. Splitting the exception handling functions in this manner facilitates quicker design times, faster cycle times, and reduced pipeline delays. According to the preferred embodiment, the performance critical exception handler includes sufficient ports to handle critical exceptions that are routed from different pipeline stages. The non-performance critical handler receives the remaining exceptions. Because these exceptions are not performance critical, this exception handler can be simplified by narrowing this exception handler to the issue width of the processor or narrower.

[0015] According to the preferred embodiment of the present invention, exceptions are classified into two categories, which comprise performance critical exceptions and non-performance critical exceptions. The performance critical exceptions may include, for example, branch mispredictions, while the non-performance critical exceptions may comprise instructions that cannot be decoded, or instructions which would produce a system error (such as a divide by zero operation). The different categories of exceptions are handled separately in two different exception handlers. Exceptions that are performance critical are handled speculatively to permit quicker resolution of these exceptions so as not to delay the program execution. Thus, performance critical exceptions are resolved even though it is not certain that these exceptions lie in the actual program path. Conversely, non-performance exceptions are handled non-speculatively, so that the exception is resolved only when it is known to lie in the actual program path.

[0016] These and other aspects of the present invention will become apparent upon analyzing the drawings, detailed description and claims, which follow.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] For a detailed description of the preferred embodiments of the invention, reference will now be made to the accompanying drawings in which:

[0018] Figure 1 is a partial illustration of a processor constructed in accordance with the preferred embodiment with two exception handlers.

NOTATION AND NOMENCLATURE

[0019] Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, processor and computer companies may refer to a component and sub-components by different names. This document does not intend to distinguish between components that differ in name but not function. In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "including, but not limited to…". Also, the term "couple" or "couples" is intended to mean either a direct or indirect electrical connection. Thus, if a first device couples to a second device, that connection may be through a direct electrical connection, or through an indirect electrical connection via other devices and connections. The term "CPU", "processor", and "microprocessor" are used synonymously to broadly refer to the device in a computer system that interprets and executes programmed instructions. The term "exception" or "excepted instruction" refers to an event or instruction that causes suspension of normal program execution. To the extent that any term is not specially defined in this specification, the intent is that the term is to be given its plain and ordinary meaning.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0020] Referring now to Figure 1, the present invention constructed in accordance with the preferred embodiment comprises a processor 50 with a plurality of pipelines 60, 70, 80 for fetching, decoding and executing program instructions. The width of the pipeline may be any arbitrary size, as denoted by the indication that *n* pipelines may be implemented. Each pipeline also has a plurality of stages at which different operations may occur, including for example, fetching, decoding and executing program instructions. Other stages may also be provided as desired in the pipeline, without limitation. Each of the pipelines have been shown as including *m* stages for purposes of illustration, with the understanding that the number of pipeline stages is not material to the teachings of the present invention.

[0021] According to the preferred embodiment, the processor 50 includes an exception handling section 100 that receives excepted instructions from one or more stages of any of the pipelines. In accordance with the preferred embodiment, and as shown in Figure 1, the exception handling section 100 comprises a speculative exception handler 150 and a non-speculative exception handler 125. The exception handlers 125, 150 may be configured as a single unit, or can be configured separately, as desired by the designer.

[0022] The speculative exception handler 150 preferably handles critical exceptions that need to be resolved in an expeditious manner so as to resume the actual program path as early as possible. An example of a critical exception in the preferred embodiment is branch mispredictions. Mispredictions occur on a relatively frequent basis, and it is generally desirable from a performance perspective to handle such exceptions on an expedited basis. A process operation cannot be completed until an exception arising from a branch misprediction is resolved. Thus, branch mispredictions potentially delay completion of a program. Other examples of performance critical exceptions (in addition to branch mispredictions) according to the preferred embodiment include load/store order traps and jump mispredictions. A load/store order trap occurs when a store instruction precedes a load address instruction in the process flow, but the load address instruction executes earlier than the store instruction. Thus, if these instructions execute out-of-order, the wrong value may be loaded. A load/store order trap preferably is recognized by one or more algorithms in the processor, and the speculative exception handler 150 then causes the load operation to be re-executed so that the proper load value is loaded in the desired register.

[0023] The exception handler 150 is referenced as a speculative handler in Figure 1, denoting that exceptions may be taken out of order, if doing so would improve performance. Thus, speculative exception handler 150 may begin processing an exception as soon as it arrives, even though prior instructions have not yet executed. Preferably, the exceptions routed to the speculative exception handler 150 are sufficiently critical to the performance of the processor 50 that the speculative exception handler 150 does not wait to confirm that the exception lies in the path of the executing program. Thus, the speculative exception handler 150 will resolve excepted instructions even though it is not clear that the exception lies in the actual proper path of the program flow. According to the preferred embodiment, the risk that time and resources may be wasted in resolving such speculative instructions is out-weighed by the performance gain that results from expeditious resolution of these critical instructions.

[0024] In accordance with the preferred embodiment, the speculative exception handler 150 preferably receives critical exceptions from each stage of the various pipelines, although this is not a requirement of the present invention. The exceptions may be detected and flagged as critical by any suitable techniques or algorithms running on the processor. Typically, the processor includes hardware that detects each possible type of exception. Thus, if the decoding engine is unable to decipher an instruction code, an invalid instruction detection algorithm will cause the instruction to be flagged and routed to the non-critical handler 125. Similarly, if a branch misprediction algorithm detects an actual program branch path that differs from the predicted path, it flags the mispredicted branch instruction and causes it to be routed it to the speculative exception handler 150. As another example, a load/store trap detector algorithm may scan each load instruction to ensure that a load/store trap has not occurred. If a load/store trap is detected, the excepted load instruction is routed to the speculative exception handler 150 with an indication that a load/store trap has occurred with this instruction. As one or more exceptions arrive at the speculative exception handler 150, the speculative exception handler 150 selects one of the excepted instructions to be resolved (which in the preferred embodiment is the oldest excepted instruction), even though the instruction may still be speculative in the sense that the processor does not yet know for sure that this instruction will need to execute. If the excepted instruction is a branch misprediction, the processor preferably will resolve the exception by initiating the fetching and decoding of the correct branch destination instruction, so that the program can be completed without excessive delay.

[0025] The non-speculative exception handler 125 preferably handles non-critical exceptions that are detected in one or more of the pipeline stages. Typically, the non-critical exceptions are those exceptions that are detected as invalid or illegal instructions. This includes instructions that cannot be decoded, and instructions which, when executed, violates a rule of operation (such as divide by zero). Other non-critical excepted instructions include arithmetic overflows and cache parity errors. These sorts of non-critical exceptions arise on a relatively infrequent basis. Because these excepted instructions are not performance critical, the non-speculative exception handler 125 does not initiate resolution of these exceptions until it is clear that they reside in the actual program path. Thus, the non-speculative exception handler 125 preferably delays resolution of the non-critical exceptions until it is clear that the instruction needs to be executed to complete the program. If the non-critical exception is discovered to be down a bad path such as from the result of a mispredicted branch, the exception is flushed without being selected by the non-speculative exception handler 125. Stated differently, the non-speculative exception handler 125 only selects exceptions for resolution that are definitely in the actual program path.

[0026] The present invention seeks to classify every possible exception into critical exceptions that need to be handled quickly for performance purposes, or non-critical exceptions that can be resolved on a less expeditious basis without significantly impacting performance. Except as specifically stated in the appended claims, the present invention is not intended to be limited to any particular classification of exceptions. Thus, the decision of which exceptions are classified in each category is arbitrary, and is left to the discretion of the system designer or architect. The net result, preferably, is that non-speculative exception handler 125 can be figured as a relatively large, but slower handler, while speculative exception handler 150 is configured as a smaller, faster handler to effectuate exception handling that enhances processor performance without a large associated cost.

[0027] In the event that multiple excepted instructions are pending, the non-speculative and speculative exception handlers 125, 150 may select the instruction to execute based on a variety of different criteria, and the criteria used between the different exception handlers may differ. Preferably, however, if multiple excepted instructions are present in either handler, the handler selects the pending instruction queued in that handler which represents the earliest excepted instruction in the program flow. In addition, if an exception arises from an invalid branch, hardware flushes the excepted instruction and all younger instructions to prevent resources from being wasted and incorrect program operation in handling an exception from an invalid branch.

[0028] The exception handlers 125, 150 may be configured in a variety of ways to recognize the existence and type of excepted instruction. For example, each of the exception handlers 125, 150 may include a queue in which excepted instructions can be stored with an appropriate flag or code that indicates the type of exception. When an instruction is selected from the queue, the instruction and flag are transmitted to appropriate logic circuitry to resolve the exception. This circuitry may be integrated with the exception handler logic, or may be separated and located elsewhere in the processor logic. Preferably, on each cycle, the speculative exception handler 150 selects an excepted instruction to resolve. Similarly, the non-speculative exception handler 125 will select an excepted instruction to resolve, if any instruction is pending in that handler that is certain to lie in the actual program path.

[0029] According to the preferred embodiment, a mutliplexer 175 or other similar circuit or logic combines the potential output from the two handlers 125, 150 onto a single set of wires to minimize the circuitry required to respond to the exception handlers 125, 150. Thus, according to the preferred embodiment, the multiplexer 175 selects one of the selected exceptions from handlers 125, 150 and signals other parts of the processor to indicate that an exception has arisen that is ripe for resolution, or which has been already resolved. As will be apparent to one skilled in the art, a variety of different multiplexing schemes may be used to determine which exceptions to select from the handlers 125, 150. Moreover, it is possible that different multiplexing decisions may be implemented in the same system, based on various operating conditions or parameters. Generally, according to the preferred embodiment, the default condition of the multiplexer 175 is to select the non-critical exceptions selected by non-speculative exception handler 125, if such an exception appears at the output port of the non-speculative exception handler 125. The non-speculative exception is chosen since it is guaranteed to be an exception from the actual program flow. If no excepted instruction is provided at the output port of the non-speculative exception handler 125, the multiplexer 175 selects the excepted instruction appearing at the output port of the speculative exception handler 150. According to one embodiment, the multiplexer 175 may be configured to periodically select the speculative handler 150 even in the event that an exception is pending at the output port of the non-speculative exception handler 125 to ensure that the speculative exception handler 150 does not become starved. As yet another alternative, various other arbitration schemes may be used to determine how the exceptions are selected from the exception handlers 125, 150, with the understanding that one of the primary criteria is that handling of critical exceptions is generally expedited, while non-speculative exceptions are guaranteed to be from the actual program path. As yet another embodiment, the multiplexer 175 may be omitted, and separate lines may extend from each exception handler 125, 150 to the rest of the processor circuitry. In this event, redundant circuitry may be required in the processor circuitry to handle the dual output ports from the exception handling section 100.

[0030] As noted above, the exception handlers 125, 150 may function to store and select exceptions as they arrive, with the resolution of the exception occurring in other logic in the processor. Thus, the output of the handlers 125, 150 may transmit the excepted instruction and type of instruction to other logic in the processor 50 that resolves the exception.

[0031] The above discussion is meant to be illustrative of the principles and various embodiments of the present invention. Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.